WE CLAIM:

1. A method to create a checkpoint for a plurality of processes in a distributed computation running on a distributed system, wherein the distributed system comprises a plurality of compute nodes, the method comprising:

suspending the inter-process communication of each of the plurality of processes;

creating a checkpoint using a checkpoint library residing on each of the compute nodes;

for each of the plurality of compute nodes:

storing a first list of memory regions modified in a checkpoint interval;

storing a second list of memory regions; and

wherein the creating of the checkpoint: (i) comprises recording the first list as the second list, (ii) write-protecting the list of memory regions modified in the checkpoint interval, and (iii) continues in parallel with computation on the each of the compute nodes until the checkpoint is completed.

2. The method of claim 1 further comprising initiating the creation of the checkpoint by sending a checkpoint command to each of the plurality of processes.

3. The method of claim 1 wherein:

the plurality of processes are executed by the plurality of compute nodes; and

the suspending of the inter-process communication of each of the plurality of processes is done for a freeze period.

4. The method of claim 3 further comprising suspending the local execution of all of the plurality of processes during the freeze period.

5. The method of claim 1 further comprising running a plurality of distributed applications on the distributed system, wherein the creating of the checkpoint improves the reliability of the distributed applications.

6. The method of claim 1 further comprising executing a distributed application and a checkpoint library on each of the plurality of compute nodes, wherein the checkpoint library is linked to the distributed application and the checkpoint library resides in the user space of the compute node.

7. The method of claim 1 wherein the first list is stored in a current commit list buffer and the second list is stored in a previous commit list buffer.

8. The method of claim 7 wherein for each of the plurality of compute nodes the creating of the checkpoint continues in parallel with computation on each of the compute nodes after the recording of the current commit list buffer and the write-protecting of the plurality of memory regions.

9 A distributed system for creating a checkpoint for a plurality of processes running on the distributed system, comprising:

a plurality of compute nodes, wherein an operating system executes on each compute node; and

a checkpoint library residing in the user space on each of the plurality of compute nodes, wherein the checkpoint library is transparent to the operating system on its corresponding one of the plurality of compute nodes; and

wherein a distributed application and middleware reside on each of the plurality of compute nodes, the checkpoint library is also transparent to the distributed application and middleware, and any one of the plurality of processes may be migrated from one node to another node within the plurality of compute nodes.

10. The distributed system of claim 9 wherein all inter-process messages, dynamic memory allocations, secondary storage access, and signal handling for each of the plurality of compute nodes are processed through the checkpoint library residing on the same compute node.

11. The distributed system of claim 9 wherein a windowed logging protocol is implemented in the checkpoint library.

12. A method of operating a distributed system in order to create a checkpoint for a plurality of processes in a distributed computation running on a distributed system, wherein the distributed system comprises a plurality of compute nodes connected by an interconnection network and the plurality of compute nodes include a sending node running a sending process and comprising a local memory and a receiving node running a receiving process and comprising a local memory, the method comprising:

committing each message of a plurality of messages sent by the sending process to a local log stored in the local memory of the sending node before sending each message to the interconnection network;

receiving each message by a checkpoint library running on the receiving node operable to pass each message to the receiving process; and

uncommitting each message from the local log of the sending node prior to passing each message to the receiving process.

13. The method of claim 12 wherein the uncommitting of each message from the local log occurs after the receipt of an acknowledgement message from the receiving process to confirm the successful delivery of each message.

14. The method of claim 12 further comprising:

creating a checkpoint for the distributed system; and

after restarting the distributed computation from the last consistent checkpoint, resending each message committed to the local log of the sending node, and not subsequently uncommitted, prior to the creating of the checkpoint.

15. The method of claim 14 further comprising:

discarding any resent message already received by the receiving process prior to creating the checkpoint; and

uncommitting any message discarded by the receiving node from the local log of the sending process.

16. A method of preemptive scheduling in which a first distributed computation comprising a plurality of processes is being executed on a distributed system, comprising:

checkpointing the first distributed computation using a windowed messaging logging protocol; and

halting execution of the first distributed computation in response to a preemptive request signal received by the distributed system; and

after the halting of the execution of the first distributed computation, resuming execution of or initiating a second distributed computation on the distributed system.

17. The method of claim 16 further comprising, after halting of the execution of the first distributed computation, migrating one or more of the plurality of processes within the distributed system.

18. The method of claim 16 wherein the preemptive request signal is generated in response to one of the following: a request for preemption from a scheduling entity coupled to the distributed system, or a request from a user of a compute node within the distributed system.

19. The method of claim 16 wherein the preemptive request signal is generated in response to the specification by a software application, having predefined knowledge of a checkpointing library executing on a compute node in the distributed system, of a checkpoint to occur at a point in execution of the distributed computation in which the amount of state required to be saved is less than a predefined quantity.

20. The method of claim 16 wherein:

the checkpointing of the first distributed computation is done periodically in a plurality of checkpoint intervals;

the preemptive request signal is generated in response to the detection of a resource failure within the distributed system; and

the checkpointing of the first distributed computation in one of the plurality of checkpoint intervals prior to the checkpoint interval in which the resource failure is detected.

21. The method of claim 16 further comprising halting execution of the second distributed computation and then resuming execution of the first distributed computation.

22. The method of claim 21 wherein the resuming of the execution of the first distributed computation is in response to a request by the scheduling entity.

23. The method of claim 16 wherein:

the checkpointing of the first distributed computation is initiated by the sending of a signal, asynchronous to the first distributed computation, from a scheduling entity; and

the halting of the execution of the first distributed computation is done after the checkpointing of the distributed computation is completed.

24. The method of claim 23 wherein the signal sent from the scheduling entity is a true operating system signal or an out-of-band message.

25. A method to store a checkpoint in a distributed system, wherein the distributed system comprises a plurality of compute nodes, the method comprising for each of the plurality of compute nodes:

maintaining a current commit list containing a list of memory regions modified in a checkpoint interval during computation on the compute node;

suspending computation on the compute node;

after suspending computation on the compute node, moving the list of memory regions contained in the current commit list to a previous commit list on the compute node;

clearing the current commit list;

write-protecting the list of memory regions modified in the checkpoint interval; and

after write-protecting the list of memory regions, resuming computation on the compute node and storing the checkpoint.

26. The method of claim 25 wherein the suspending of computation on the compute node comprises suspending of interprocess communication.

27. The method of claim 25 wherein the storing of the checkpoint comprises:

storing each of the memory regions listed in the previous commit list to a checkpoint file; and

after a memory region is stored to the checkpoint file, removing the memory region from the previous commit list.

28. The method of claim 27 wherein the storing of the memory regions listed in the previous commit list comprises traversing the memory regions in the order listed in the previous commit list.

29. The method of claim 25 wherein the write-protecting of the list of memory regions modified in the checkpoint interval is done after the moving of the list of memory regions contained in the current commit list to the previous commit list.

30. The method of claim 25 wherein the write-protecting of the list of memory regions modified in the checkpoint interval is done prior to the moving of the list of memory regions contained in the current commit list to the previous commit list.

31. The method of claim 25 further comprising storing the checkpoint to non-volatile storage after the resuming of computation.

32. The method of claim 31 wherein the non-volatile storage is a shared storage facility accessible by each of the plurality of compute nodes.

33. The method of claim 31 wherein:

the storing of the checkpoint comprises storing information to a checkpoint file; and

further comprising in response to an attempt by the computation on the compute node to access any write-protected memory region on the compute node, if the write-protected memory region is on the previous commit list:

writing the write-protected memory region to the checkpoint file; and

removing the write-protected memory region from the previous commit list.

34. The method of claim 33 further comprising, after the attempt to access any write-protected memory region on the previous commit list, adding the write-protected memory region to the current commit list.

35. The method of claim 31 wherein the storing of the checkpoint comprises storing information to a checkpoint file and further comprising in response to an attempt by the computation on the compute node to access any write-protected memory region on the compute node:

interrupting the storing of the checkpoint and computation on the compute node, and determining if the attempt to access the write-protected memory region includes any memory region listed in the previous commit list;

if the attempt to access does not include any memory region listed in the previous commit list, marking the write-protected memory region as being modified in the current commit list and removing the write protection of the memory region; and

if the attempt to access includes any memory region listed in the previous commit list, writing the listed memory region to the checkpoint file, removing the memory region from the previous commit list, marking the write-protected memory region as being modified in the current commit list and removing the write protection of the memory region.

36. The method of claim 35 further comprising, after removing the write protection of the memory region, resuming the storing of the checkpoint and computation on the compute node.

37. The method of claim 35 wherein the write-protected memory region is any memory region addressable by the compute node.

38. A method of migrating a process, taking part as one of many processes performing a distributed computation in a distributed system, from a first compute node to a second compute node in the distributed system, comprising:

checkpointing the distributed computation using a windowed messaging logging protocol; and

migrating the process from the first compute node to the second compute node.

39. The method of claim 38 wherein the checkpointing comprises running a checkpoint library at the user level on the first compute node and redirecting all inter-process communication on the first compute node through the checkpoint library in order to implement the logging protocol.

40. The method of claim 39 wherein the migrating comprises redirecting all communication intended for the process to the second compute node.

41. The method of claim 40 wherein the redirecting of all communication comprises using the checkpoint library to map a plurality of hardware addresses of the first compute node to corresponding hardware addresses of the second compute node.

42. The method of claim 38 wherein the migrating of the process is performed in response to a failure of the first compute node that interrupts the execution of the process on the first compute node.

43. The method of claim 38 wherein the migrating of the process is performed in response to a request from a system administrator to remove one or more compute nodes, including the first compute node, from the distributed system without substantially disrupting the distributed computation.

44. The method of claim 43 wherein the removal of the one or more compute nodes is done to permit maintenance or upgrades to the one or more compute nodes.

45. The method of claim 43 further comprising reserving a set of one or more spare compute nodes in the distributed system, wherein the second compute node is included in the set of one or more spare compute nodes.

46. The method of claim 38 further comprising:

managing the resources of the distributed system using a scheduling entity that is executing a preemptive scheduling algorithm;

prior to migrating the process, halting execution of the distributed computation in order to use the first compute node for a different distributed computation; and

wherein the migrating of the process is performed in response to a request from the intelligent scheduling entity.